SIAM J. SCI. COMPUT.
Vol. 22, No. 1, pp. 95–112

A PARALLEL JACOBI–DAVIDSON-TYPE METHOD FOR SOLVING LARGE GENERALIZED EIGENVALUE PROBLEMS IN MAGNETOHYDRODYNAMICS*

MARGREET NOOL† AND AUKE VAN DER PLOEG‡

Abstract. We study the solution of generalized eigenproblems generated by a model which is used for stability investigation of tokamak plasmas. The eigenvalue problems are of the form $Ax = \lambda Bx$, in which the complex matrices A and B are block-tridiagonal, and B is Hermitian positive definite. The Jacobi–Davidson method appears to be an excellent method for parallel computation of a few selected eigenvalues because the basic ingredients are matrix vector products, vector updates, and inner products. The method is based on solving projected eigenproblems of order typically less than 30.

We apply a complete block LU decomposition in which reordering strategies based on a combination of block cyclic reduction and domain decomposition result in a well-parallelizable algorithm. One decomposition can be used for the calculation of several eigenvalues. Spectral transformations are presented to compute certain interior eigenvalues and their associated eigenvectors. The convergence behavior of several variants of the Jacobi–Davidson algorithm is examined. Special attention is paid to the parallel performance, memory requirements, and prediction of the speed-up. Numerical results obtained on a distributed memory Cray T3E are shown.

Key words. generalized eigenvalue problems, Jacobi–Davidson method, Arnoldi method, block-tridiagonal systems, parallelization

AMS subject classifications. Primary, 65-04, 65F10, 65F15, 65F50, 65N25; Secondary, 65Y05, 65Y20

PII. S106482759933290X

1. Introduction.

Consider the generalized eigenvalue problem

$$Ax = \lambda Bx, \qquad A, B \in \mathbb{C}^{N_t \times N_t}, \tag{1}$$

where A and B are complex block-tridiagonal $N_t \times N_t$ matrices and B is Hermitian positive definite. The number of diagonal blocks is denoted by N and the blocks are $n \times n$; therefore $N_t = N \times n$.

Eigenvalue problems arise in many applications. We are particularly interested in generalized eigenvalue problems as they occur in linear magnetohydrodynamics (MHD). Such problems are generated by a finite-element spectral code CASTOR (complex Alfvén spectrum of toroidal plasmas) [8]. This code is applied intensively at the FOM Institute voor Plasmafysica, Nieuwegein (the Netherlands) for the stability investigation of tokamak plasmas [9, 13]. The physicists are particularly interested in accurate approximations of certain interior eigenvalues, called the *Alfvén spectrum* and their associated eigenvectors. Figure 1 shows the complete and the Alfvén spectrum of (1) for a small test problem with $N = 50$ and $n = 32$. A target σ is given in the neighborhood in which we want to find several eigenvalues with corresponding eigenvectors. In general, the subblocks of A are dense and N_t can be very large,

*Received by the editors January 18, 1999; accepted for publication (in revised form) April 13, 1999; published electronically June 13, 2000. This research was sponsored partly by the Cray Research Grants Program (project CRG 96.14) of NCF (Dutch National Computing Facilities Foundation) and by the Priority Program "Massaal Parallel Rekenen" (project 95MPR04) of NWO (Dutch Organization for Scientific Research).

http://www.siam.org/journals/sisc/22-1/33290.html

†CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (Margreet.Nool@cwi.nl).

‡MARIN, P.O. Box 28, 6700 AA Wageningen, The Netherlands (A.v.d.Ploeg@marin.nl).

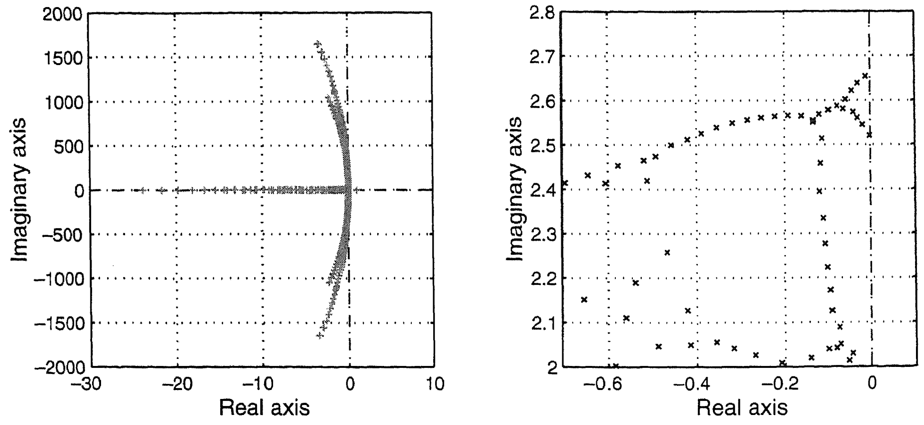

FIG. 1. *Entire (left) and Alfvén (right) spectrum of a small MHD problem:* $N = 50$, $n = 32$.

so computer storage demands are very high. Until now powerful shared memory machines with a huge amount of memory, such as the Cray C90, were used for this purpose. In this paper, we study as an alternative the feasibility of parallel computers with a large distributed memory for solving large generalized eigenvalue problems.

A promising method for computing selected eigenvalues of (1) is the Jacobi–Davidson (JD) algorithm [16, 3, 4, 5, 15]. The method has become very popular, and since we have started this project many aspects have been investigated, for instance, a variant with harmonic Ritz values has already been implemented [12]. In the future, we consider implementation of the Jacobi–Davidson QZ algorithm (JDQZ) [6] and inclusion of explicit deflation. In section 2 we briefly describe the JD algorithm for solving a block-tridiagonal eigenvalue problem. Within JD a parallel method to compute the action of the inverse of the block-tridiagonal matrix $A - \sigma B$ is used. In this approach, called DDCR, a block-reordering based on a combination of domain decomposition and cyclic reduction is combined with a complete block-tridiagonal LU decomposition of $A - \sigma B$, so $LU = A - \sigma B$. Both the construction of L and U and the triangular solves parallelize well. In this report we concentrate on the application of DDCR within JD. Originally, it was developed as a parallel preconditioner to apply within JD and appropriate for solving MHD eigenvalue problems. Some aspects of this technique are discussed in section 3.

However, the availability of a complete LU decomposition gives us the opportunity to apply the Jacobi–Davidson method to a *standard* eigenvalue problem instead of a *generalized* eigenvalue problem. Writing (1) as $(A - \sigma B)x = (\lambda - \sigma)Bx$, we have

$$x = (\lambda - \sigma)(A - \sigma B)^{-1}Bx.$$

If we define Q as $(A - \sigma B)^{-1}B$, for some value σ, we obtain the standard eigenvalue problem

$$Qx = \mu x \quad \text{with } \mu = \frac{1}{\lambda - \sigma} \Leftrightarrow \lambda = \sigma + \frac{1}{\mu}. \tag{2}$$

The eigenvalues μ of (2) form the dominant part of the spectrum, which makes them relatively easy to compute. In practice, one has to be careful because small pivot elements can be generated during the decomposition of $A - \sigma B$, especially when σ is very close to an eigenvalue. In that case, taking $Q = (LU)^{-1}B$ in (2) may influence the computed spectrum. It is advisable to visualize the obtained eigenvalue spectrum and to compare it with results of a run with another target in the neighborhood of the spectrum. A smaller tolerance can also give information about the accuracy.

As a basis for the development of the parallel code, we have taken a sequential FORTRAN code for the JD algorithm, which was developed by the late Albert Booten at CWI [3, 4, 5]. The data structure for storing the matrices was modified in such a way that we could use optimized BLAS routines as much as possible. Furthermore, that part of the CASTOR code [8] which generates the matrices A and B has been parallelized and coupled with the JD code. Of course, other (generalized) eigenvalue problems satisfying (1) can be solved by our code. Several subroutines are available to read data efficiently from files and to distribute them to the processors. Section 4 describes the data organization of the JD code on the Cray T3E. Moreover, the memory requirements are given for both our sequential and parallel implementation.

In section 5 the convergence behavior of several variants of the Jacobi–Davidson algorithm is examined. Numerical results for three MHD eigenproblems obtained by a Cray T3E are presented and analyzed, and an analysis for the parallel speed-up is given. Conclusions are drawn in section 6.

2. The JD algorithm.

Recently the JD method has been introduced as a new powerful technique for solving a variety of eigenproblems [16]. In this section we describe the method briefly and comment on our implementation. Suppose an eigenvector x is approximated by a linear combination of k search vectors $v_j, j = 1, 2, \dots, k$, where k is very small compared with N_t. Let V_k be the $N_t \times k$ matrix whose columns are given by v_j. Then the approximation to the eigenvector can be written as $V_k s$, for some k-vector s. The search directions v_j are made orthonormal to each other, hence $V_k^* V_k = I$.

Suppose that an approximation to an eigenvalue is denoted by θ. The vector s and the scalar θ are constructed in such a way that the vector $r = QV_k s - \theta V_k s$ is orthogonal to the k search directions. From this Rayleigh–Ritz requirement it follows that

$$V_k^* Q V_k s = \theta V_k^* V_k s \iff V_k^* Q V_k s = \theta s. \tag{3}$$

In this way one obtains a "projected" eigenvalue problem in which the order of the matrix $V_k^* Q V_k$ is k. By using a proper restart technique, k can remain so small that this problem can be solved by the QR method. The approximate eigenvalue θ is the eigenvalue of the projected system with the largest modulus.

At each step of the algorithm, a new search direction is constructed. Suppose that we have obtained an approximation $u = V_k s$ of the *true* eigenvector x associated with some eigenvalue μ. We assume that $\|u\|_2 = 1$, hence $\theta = u^* Q u$ is an approximation of μ. Let us define $P = uu^*$ as the orthogonal projector onto the subspace spanned by $\{u\}$. Then $I - P$ is the projector onto the orthogonal complement of span$\{u\}$, which is denoted by $u^\perp$. Any vector $x \in \mathbb{C}^n$ can be written as $x = x_1 + x_2$ with $x_1 \in$ span$\{u\}$ and $x_2 \in u^\perp$. We can scale x such that $x = u + z$ with $z \perp u$. In the JD algorithm a correction vector $z \in u^\perp$ is constructed. The restriction of Q to $u^\perp$ is given by

$$Q_{\mathrm{P}} = (I - P) Q (I - P). \tag{4}$$

If we rewrite (4) and substitute the resulting expression for Q into $Qx = \mu x$, we

```
*Jacobi-Davidson for* $Qx = \mu x, Q = (A - \sigma B)^{-1}B$.
*Parameters:* $iter, N_{ev}, tol_{JD}, k_{min}, m \ (m \geq k_{min} + N_{ev}), it_{SOL}$.
step 0: initialize
    Choose an initial vector $v_1$ with $\|v_1\|_2 = 1$; set $V_1 = [v_1]$;
    $W_1 = [Qv_1]$; $k = 1$; $it = 1$; $n_{ev} = 0$
step 1: update the projected system
    Compute the last column and row of $H_k := V_k^* W_k$
step 2: solve and choose approximate eigensolution of projected system
    Compute the eigenvalues $\theta_1, \ldots, \theta_k$ of $H_k$ and choose $\theta := \theta_j$ with $|\theta_j|$ maximal
    and $\theta_j \neq \mu_i$, for $i = 1, \ldots, n_{ev}$; compute associated eigenvector $s$ with $\|s\|_2 = 1$
step 3: compute Ritz vector and check accuracy
    Let $u$ be the Ritz vector $V_k s$; compute the vector $r := W_k s - \theta u$;
    if the residual $\hat{r} := (A - (\sigma + \frac{1}{\theta})B)u$ satisfies $\|\hat{r}\|_2 / |\sigma + \frac{1}{\theta}| < tol_{JD}$ then
        $n_{ev} := n_{ev} + 1$; $\mu_{n_{ev}} := \theta$; if $n_{ev} = N_{ev}$ stop; goto 2
    else if $it = iter$ stop
    end if
step 4: solve correction equation approximately with $it_{SOL}$ steps of GMRES
    Determine an approximate solution $\tilde{z}$ of $z$ in
    $(I - uu^*)(Q - \theta I)(I - uu^*)z = -r \ \wedge \ u^* z = 0$
step 5: restart if projected system has reached its maximum order
    if $k = m$ then
        5a: set $k = k_{min} + n_{ev}$; compute the $k$ eigenvalues of $H_m$ with largest modulus;
            construct $C \in \mathbb{C}^{m \times k}$ with columns the associated eigenvectors;
            orthonormalize columns of $C$; compute $H_k := C^* H_m C$
        5b: compute $V_k := V_m C$; $W_k := W_m C$
    end if
step 6: add new search direction
    $k := k + 1$; $it := it + 1$; call $MGS\ [V_{k-1}, \tilde{z}]$; set $V_k = [V_{k-1}, \tilde{z}]$; $W_k = [W_{k-1}, Q\tilde{z}]$;
    goto 1
```

FIG. 2. *JD algorithm after spectral transformation.*

obtain, using $Qu - \theta u = r$, $z \perp u$, $Pu = u$, and $Pz = 0$,

$$(Q_P - \mu I)z = -r + (\mu - \theta - u^* Qz)u. \tag{5}$$

Since r is orthogonal to u, premultiplication of (5) by u^* yields $\mu = \theta + u^* Qz$. Note that μ is unknown and its best approximation will be θ. In this way, we obtain as a correction equation for z

$$(I - P)(Q - \theta I)(I - P)z = -r, \qquad u^* z = 0. \tag{6}$$

It is sufficient to solve (6) only approximately. This can be done by some steps of an iterative method, for example, GMRES [14]. When an approximate solution $\tilde{z}$ of (6) has been constructed, it is made orthogonal to the previous search directions, and the new search vector v_{k+1} is taken equal to $\tilde{z}/\|\tilde{z}\|_2$. Then k is increased by 1, and the new matrix $V_k^* Q V_k$ is constructed by expanding the "old" matrix by one new row and one new column.

Our implementation of the JD method for the computation of *several* eigenvalues is shown in Figure 2. The following remarks apply to the algorithm:

(i) The number of iteration steps that have been performed is denoted by *it*. The maximum allowed number of iterations is equal to *iter*.

(ii) The method is accelerated by searching for the best eigenvector approximation by selecting θ such that $|\theta|$ is maximal. To include the possibility of multiple eigenvalues, the condition $\theta_j \neq \mu_i, j = 1, \ldots, k$ must be interpreted as follows: an accepted eigenvalue μ_i can be equal to only one θ_j. So, if the projected system contains more than one θ_j close to μ_i, then it may be a multiple eigenvalue and, since its modulus is maximal, it becomes the next selected eigenvalue.

(iii) The value n_{ev} indicates the number of eigenvalues found so far that satisfy the acceptance criterion, and the parameter N_{ev} is the number of eigenvalues that we are looking for. The approximate eigenvalues θ that satisfy the acceptance criterion

$$\frac{\|\hat{r}\|_2}{|\sigma + \frac{1}{\theta}|} < tol_{JD} \quad \text{with } \hat{r} := (A - (\sigma + \tfrac{1}{\theta})B)u, \tag{7}$$

are referred to as μ_i for $i = 1, \ldots, n_{ev}$. The algorithm stops when n_{ev} is equal to N_{ev} or when *it* equals *iter*. In order to reduce computation time, we compute the actual residual $\hat{r}$ only if r satisfies

$$\frac{\|r\|_2}{|\theta|} < 100 \times tol_{JD} \quad \text{with } r := W_k s - \theta u = (Q - \theta I)u. \tag{8}$$

From experiments we observe that (7) is hardly satisfied when (8) is violated.

(iv) In the actual implementation of the algorithm, precautions have to be taken in step 2: theoretically, it is possible that all eigensolutions of the projected system satisfy the acceptance criterion. If that would happen, no new θ and corresponding approximate eigenvector u can be found in step 2. In that case, a vector $\tilde{z}$ is chosen that is not in the subspace spanned by V_k, and the algorithm proceeds at step 5.

(v) The subspace spanned by V_k contains the eigenvectors corresponding to the eigenvalues found before, together with some search directions. In this way, implicit deflation is incorporated automatically, which means that the detected eigenvectors are not filtered out. Of course, they may not influence the progress in finding the next eigenpair or lead to an eigenpair already found. The process of acceleration to find the next eigenpair, as described above, ensures that each accepted eigenpair will be unique. This technique differs from the deflation technique described in [6] which uses explicit deflation. The latter technique can be more stable but requires more operations per iteration step.

(vi) We require that the number of search directions k may not become too large. Therefore, if k has reached the value m, the upper bound for the order of the projected eigenvalue problem, k is reduced to $k_{min}+n_{ev}$. This is accomplished by extracting the most interesting information from H_k: the vectors which lead to previously accepted eigenvectors of the original problem are kept in the system together with those vectors that correspond to the k_{min} most promising eigenvalues, with $|\theta|$ maximal. In other words, $m-(k_{min}+n_{ev})$ vectors are thrown away. The columns of the remaining matrix $C \in \mathbb{C}^{m\times k}$ are again orthonormalized and a new H_k is obtained by $H_k := C^* H_m C$. Postmultiplication of V_m and W_m with C provides the new bases V_k and W_k. This restart technique is also applied in [6].

(vii) We use modified Gram-Schmidt (MGS) [7] to orthonormalize vectors;"call *MGS* $[V_{k-1}, \tilde{z}]$" in step 6 means that $\tilde{z}$ is made orthonormal to the columns of V_{k-1}. As suggested in [16] we apply MGS twice. Although classical Gram-Schmidt (CGS) has better parallel properties, we use MGS because it is more stable. Evidently, all inner products and all vector updates in our parallel implementation have been parallelized.

(viii) The original code that was taken as a basis for this project did not use harmonic Ritz values [6, 16]. Numerical experiments with a comparable algorithm using harmonic Ritz values have been described in [12]. The main advantage is that the LU decomposition of $A - \sigma B$ is used as a preconditioner and not as a shift and invert technique; then rounding-off errors in the decomposition for very large eigenvalue problems have less influence on the eigenvalue spectrum. We mention that the harmonic approach requires more memory and costs about 20% more execution time per Jacobi–Davidson iteration step.

2.1. Solving the correction equation. In step 4 of the JD algorithm (Figure 2), we must solve the correction equation

$$(I - uu^*)(Q - \theta I)(I - uu^*)z = -r \quad \text{and} \quad u^*z = 0 \tag{9}$$

approximately. From $u^*z = 0$, (9) can be rewritten as $(I - uu^*)(Q - \theta I)z = -r$, which is equivalent to $(Q - \theta I)z + \varepsilon u = -r$ with $\varepsilon = -u^*(Q - \theta I)z$ (cf. [16]). In order to construct a suitable preconditioner for (9), it may be useful to write the equation in the augmented form of order $N_t + 1$,

$$\begin{bmatrix} Q - \theta I & u \\ u^* & 0 \end{bmatrix} \begin{bmatrix} z \\ \varepsilon \end{bmatrix} = \begin{bmatrix} -r \\ 0 \end{bmatrix} \tag{10}$$

(cf. also [15, Theorem 3.5]). If $Q - \theta I$ is nearly singular, then the matrix will be ill-conditioned; the augmented form leads to a better conditioned matrix. The augmented correction equation can be solved, for instance, by applying some fixed number of steps of GMRES(m) (at most m steps with full GMRES, no restarts). A special choice for m is 0 (or $it_{SOL} = 0$), in which case the JD algorithm is closely related to the Arnoldi method [7]. Important differences between the JD algorithm of Figure 2 with $it_{SOL} = 0$ and the Arnoldi method are the structure of the matrix H_k and the way the restarts are performed. The projected matrices H_k obtained by applying the Arnoldi factorization are always upper Hessenberg with nonnegative subdiagonal elements. The matrices H_k obtained by k steps of JD do not have this structure (except for $it_{SOL} = 0$ and $k \leq m$, although rounding errors may appear for growing k). So, the computation of the eigenvalues of H_k requires an extra step to transform H_k to upper Hessenberg. Since JD does not have to repair the Hessenberg form, the restart technique in step 5 can be kept simple (see section 2). For details on the restart technique in the implicitly restarted Arnoldi method (IRAM), in which the Hessenberg structure of H_k has to be retained, we refer to [10]. A third difference with Arnoldi is the choice of θ; by selecting $|\theta|$ maximal the JD method is accelerated. We also refer to Discussion 4.1 in [15] for a clear discussion on the relation between the JD method, including the exact solution of the correction equation and shift-and-inverse Arnoldi. In that discussion no restarts are considered.

The convergence of GMRES(m) may be accelerated by incorporating a suitable (cheap) preconditioner. As in Booten et al. [5] and Sleijpen et al. [15], we rewrite the inverse of the augmented matrix, with $K = Q - \theta I$, as

$$\begin{bmatrix} K & u \\ u^* & 0 \end{bmatrix}^{-1} = \begin{bmatrix} I & -K^{-1}u \\ 0^* & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ u^*/\nu & -1/\nu \end{bmatrix} \begin{bmatrix} K^{-1} & 0 \\ 0^* & 1 \end{bmatrix} \tag{11}$$

with $\nu = u^*K^{-1}u$. It is easy to verify that

$$K = Q - \theta I = -\theta(A - \sigma B)^{-1}[(A - (\sigma + \tfrac{1}{\theta}B)] \tag{12}$$

and K^{-1} may be approximated by $-\frac{1}{\theta}I$. Hence, by replacing K^{-1} with $-\frac{1}{\theta}I$ in (11) and ν by $-\frac{1}{\theta}u^*u = -\frac{1}{\theta}$, we obtain the preconditioner $\mathcal{M}^{-1}$:

$$\begin{bmatrix} K & u \\ u^* & 0 \end{bmatrix}^{-1} \approx \begin{bmatrix} -\frac{1}{\theta}(I - uu^*) & u \\ u^* & \theta \end{bmatrix} = \mathcal{M}^{-1}. \tag{13}$$

The augmented equation (10) can be preconditioned by $\mathcal{M}^{-1}$ and then in almost all cases the number of JD iterations needed to find the first eigenvalue will be smaller than in cases where no preconditioning is used, as numerical experiments demonstrate in section 5.

3. The LU decomposition.

To calculate an Alfvén spectrum we choose some target values σ in the neighborhood of this spectrum. For each σ we compute a complete LU decomposition of $A - \sigma B$. One (L, U) pair can be used for the calculation of several eigenvalues. However, if the method becomes too slow, it might be useful to choose a new target σ.

3.1. Block-tridiagonal LU approach.

The matrices A and B in (1) have the same block-tridiagonal structure and therefore $A - \sigma B$ is also block-tridiagonal. Since the subblocks are more than 50% full, a complete LU decomposition of $A - \sigma B$ is relatively cheap: the total number of nonzeros in $L + U$ is at most twice as much as in $A - \sigma B$. In the rest of this paper, this technique is referred to as the block-tridiagonal LU approach. The decomposition is performed on a block level. This enables us to use partial pivoting, which makes the decomposition more robust. In order not to disturb the block-tridiagonal structure, the search for pivot elements is restricted to the blocks on the main diagonal. In our experiments, this pivot strategy seems to work well.

For a large number of diagonal blocks, the total number of complex multiplications required for the construction of L and U is approximately $\frac{7}{3}\,Nn^3$, and the number of multiplications required for performing the triangular solves with both L and U once is approximately $3Nn^2$.

3.2. The DDCR method.

To improve parallelization possibilities of block-tridiagonal LU decomposition, we use a reordering based on a combination of domain decomposition (DD) and cyclic reduction (CR), discussed in [18].

Let p be the number of processors that is actually used, and let the integer $N_p = \lceil \frac{N}{p} \rceil$ [1] represent the number of diagonal blocks to be treated on each processor (except possibly the last processor, on which the number of diagonal blocks can be less). The first step of the DDCR method is to perform a block-reordering of both block rows and block columns based on a domain decomposition strategy with p nonoverlapping subdomains. The second step of the DDCR method is to construct a block lower-triangular matrix L and a block upper-triangular matrix U in such a way that $A - \sigma B = LU$ and all blocks on the main diagonal of U are identity matrices.

For N_p large, the construction of a block-tridiagonal LU decomposition of the first block-tridiagonal matrix on the first processor costs about $\frac{7}{3}\,n^3N_p$ multiplications. In [18] it is shown that the rest of the block-tridiagonal LU decomposition costs about $\frac{19}{3}\,n^3(N - N_p)$ multiplications. Hence the total number of multiplications required for the construction of L and U is approximately

$$\left(\tfrac{7}{3}\,N_p + \tfrac{19}{3}\,(N - N_p)\right)n^3 = \tfrac{1}{3}\,(19N - 12N_p)n^3. \tag{14}$$

[1]By $\lceil x \rceil$ we denote the smallest integer $\geq x$ and by $\lfloor x \rfloor$ the largest integer $\leq x$.

In the same way it can be shown that for large N_p the number of multiplications required for performing the triangular solves with both L and U once is approximately

$$(3N_p + 5(N - N_p))n^2 = (5N - 2N_p)n^2. \tag{15}$$

4. The parallel implementation and data organization. We started this project with the parallelization of the original code [3, 4, 5] on a Cray T3D in Eagan, MN. The last part of the development has been done on a Cray T3E in Delft, the Netherlands. Some important characteristics of the Cray T3E in Delft are listed in Table 1.

TABLE 1
Main characteristics of the Cray T3E at HPαC centre, Delft.

# processors	80 (72 in parallel)
Clock period	3.33 ns clock
Peak performance / processor	600 Mflop/s
Total peak performance	33.6 Gflop/s
Local memory	128 Mbytes
Total memory	10 Gbytes
Data cache size	8 Kbytes
Compiling system	Cray CF90
GiGaRing channels	available

The communication steps in the recent code have been implemented with fast `SHMEM` routines (shared memory access library). These `SHMEM` routines are data passing library routines similar to message passing library routines and minimize the overhead associated with data passing requests, maximize bandwidth, and minimize data latency. Optimized `BLAS` routines are used as much as possible to achieve high performance per processor, and for the linear algebra part LAPACK routines [1] are called. For considerations on the programming technique, data distribution and communication on the MHD application, we refer to [11].

4.1. Distribution of the MHD matrices. For large problems with a lot of data, the I/O is often very expensive. Therefore, we do not read the MHD matrices from the files but generate them during execution, which takes only a few seconds. A part of the CASTOR code, which computes the matrix elements, has been coupled with our JD code. Moreover, it has been parallelized such that the available processors generate their own pieces of the matrices A and B in (1). The matrices are distributed block-row-wise along the processors; each processor gets one or more main diagonal blocks with their nearest sub- and superdiagonal blocks belonging to the same block row. All vectors are distributed accordingly. As a consequence, for a matrix-vector multiplication the memory traffic (of only two vectors of length n) is reduced to the first sub- and last superdiagonal block per processor.

4.2. Data format: CRS format versus dense block-tridiagonal format. The compressed row storage (CRS) format puts the subsequent nonzeros of the matrix rows in contiguous memory locations. To describe a sparse matrix A, we need three vectors, one for complex nonzero floating point numbers (*mat_A*) and two for integers (*colind_A*, *rowptr_A*). For more details and examples, we refer to [2].

CRS format has the advantage that it can be used to store general sparse matrices. However, the DDCR decomposition is completely based on the block-tridiagonal form of $A - \sigma B$. This implies that, for our applications, CRS format can be advantageous only if the blocks of A and B are sparse enough. From Table 2, which gives the

TABLE 2
Times for the matrix-vector multiplication on a single processor of a Cray T3E. Results for the block-tridiagonal form (dense blocks including zero values) are denoted by DENSE. *Results for CRS format are denoted by* CRS(*A*) *and* CRS(*B*) *for A and B, respectively.*

Execution times in μ-seconds on a Cray T3E, $N = 40$.					
MATVEC	$n = 16$	$n = 32$	$n = 48$	$n = 64$	$n = 80$
DENSE BLOCKS	4490	14577	32055	54666	93924
CRS(A)	2469	9874	22518	33733	52861
CRS(B)	1421	4979	10806	15814	24374
Number of nonzeros in A and B.					
$nnz(A)$	11392	47906	109542	196300	308180
$nnz(B)$	5934	22778	50532	89196	138770

execution times for matrix-vector multiplications on a single processor of a Cray T3E for matrices A and B generated by the CASTOR code, we may conclude that CRS format is to be preferred for both A and B on this platform.

4.3. Memory requirements. In this section, we will indicate which amount of memory is required for both the sequential and parallel implementation, expressed in $nnz(A)$ and $nnz(B)$, the numbers of nonzero values in the A and B, respectively, and N, n, and N_{ev}, where N denotes the number of diagonal blocks, n the blocksize of the block-tridiagonal matrices, and N_{ev} the number of eigenvalues we are looking for. The last parameter is m, the maximum allowed value of the projected system. As stated in section 2, m is much smaller than the order $N \times n$ of the matrices A and B.

A CRS-formatted matrix of $nnz(A)$ nonzero values requires 16 $nnz(A)$ bytes for storing mat_A and 8 $nnz(A)$ bytes for $colind\_A$, respectively (cf. section 4.2). The integer array $rowptr\_A$ uses an additional 8 Nn bytes. The memory space for the CRS-formatted matrix B can be expressed in $nnz(B)$, analogously. The block-tridiagonal LU decomposition needs 3 Nn^2 COMPLEX words. The subspace matrix V_k and associated space W_k use 2 Nnm COMPLEX words, and the restart in step 5 of Figure 2 requires (in the worst case) another Nnm COMPLEX words. The total requirements for those matrices are 3 Nnm COMPLEX words. The accepted eigenvectors are stored in a matrix of order $Nn \times N_{ev}$. An additional working space for some vectors takes 5 Nn COMPLEX words.

Besides the matrix H_k, its Hessenberg form and its eigenvectors plus some working space require $4m(m+1)$ COMPLEX words. Summarizing, the amount of memory required for the current *sequential* implementation in complex arithmetic, where one COMPLEX word corresponds with 16 bytes, is approximately

$$24\,(nnz(A) + nnz(B)) + 16Nn(3(n+m) + N_{ev} + 7) + 64m(m+1) \text{ bytes.} \tag{16}$$

For the MHD matrices, the average sparsity of the sub-, super-, and diagonal blocks of A is approximately 41% and of B 20%, i.e., $nnz(A) \approx 0.41 \times 3Nn^2$ and $nnz(B) \approx 0.20 \times 3Nn^2$. Thus, the amount of memory for the sequential implementation (16) approximates

$$16Nn(5.75n + 3m + N_{ev} + 7) + 64m(m+1) \text{ bytes.} \tag{17}$$

The parallel implementation requires per processor approximately an additional $32n(N_pn+3n)$ bytes for the DDCR solver and some extra memory (10 %) to distribute the CRS matrices, in case their sparsity is not equally partitioned over the block

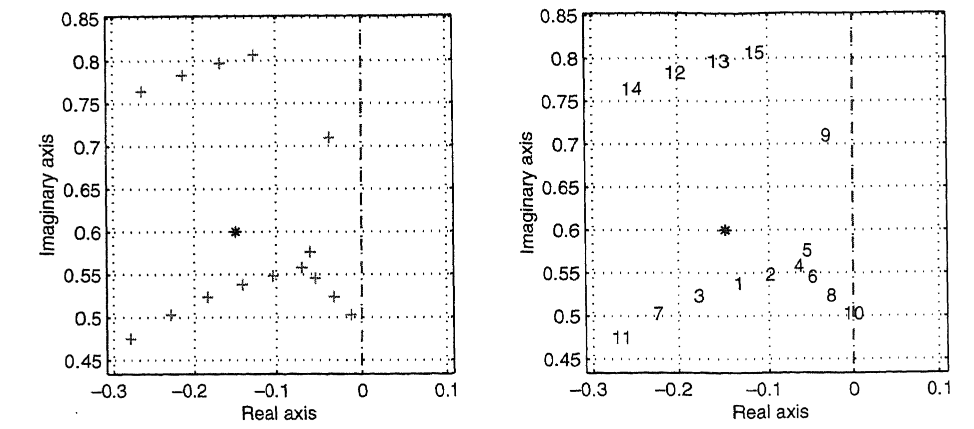

FIG. 3. *Part of the eigenvalue spectrum of an MHD problem with $N = 40, n = 64$. The target σ is indicated by an "$*$". In the right picture the order in which they appear are shown.*

rows. The small matrix of the projected system is present on all processors, and sequential steps, like the solution of the projected eigenvalue problem, are performed by all processors to avoid communication. Summarizing, the amount of memory (per processor) for our *parallel* implementation approximates

$$(18)\quad \begin{aligned} &48N_pn^2 + 16N_pn(3(n+m) + N_{ev} + 7) + 32n(N_pn + 3n) + 64m(m+1) \text{ bytes} \\ &= 128N_pn^2 + 16N_p(3m + N_{ev} + 7) + 96n^2 + 64m(m+1) \text{ bytes.} \end{aligned}$$

5. Numerical results.

5.1. The influence of the correction equation. In the left picture of Figure 3, the eigenvalues of an MHD problem with $N = 40$ and $n = 64$ near the target $\sigma = (-0.15, 0.60)$ are plotted. In the right picture, the eigenvalues are numbered consecutively as they are found. The solid line in Figure 4 displays the convergence behavior of this MHD problem solved by JD with $it_{SOL} = 0$. Each "o" indicates a converged eigenvalue. Obviously, to find the first eigenvalue is rather expensive, but after that only a few iteration steps are required to find the next ones. Note that after 30 and 39 JD iteration steps two eigenvalues were even found in one step. It appears that with this target even more eigenvalues (probably of the upper Alfvén branch) can be found than the 15 we asked for. However, since an implicit restarting technique is used, the order of the projected system will then grow . A better solution will be to choose a new target based on the results of the last (few) runs.

The question arises if it is possible to reduce the number of JD iteration steps when GMRES(m) is applied to the augmented correction equation (10). In Table 3, we distinguish four variants to solve the correction equation:

- JD-sol: in each JD step GMRES(m) is applied *without* preconditioning.
- JD-sol*: GMRES(m) is applied only if $\|\hat{r}\|_2 < \varepsilon_{tr}$ *without* preconditioning.
- JD-solP: in each JD step GMRES(m) is applied *with* preconditioning from the left by $\mathcal{M}^{-1}$.
- JD-solP*: GMRES(m) is applied only if $\|\hat{r}\|_2 < \varepsilon_{tr}$ *with* preconditioning from the left by $\mathcal{M}^{-1}$.

The preconditioning matrix $\mathcal{M}^{-1}$ has been introduced in section 2.1. The JD-sol variant does not improve the convergence process: from Figure 4 we know that if $it_{SOL} = 0$, 23 iterations are needed to get the first eigenvalue. It is known that in the

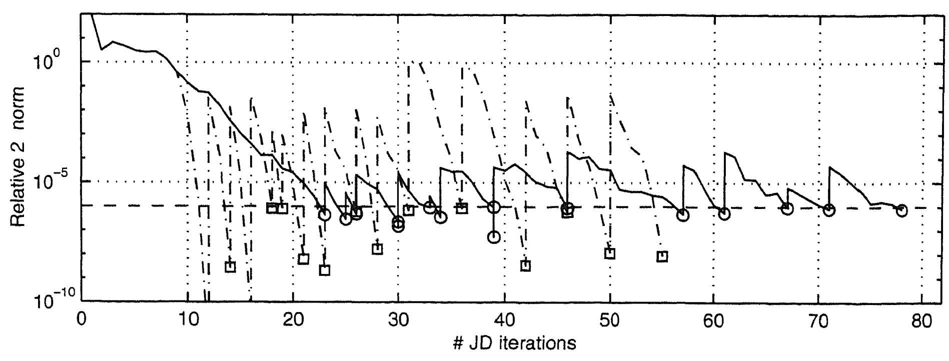

FIG. 4. *Convergence behavior of JD with* it_{SOL} = 0 *(solid line with "○"s) and* JD-solP*, GMRES(20) *(dashed line with "□"s);* $N = 40, n = 64$.

TABLE 3

Both the number of JD iterations as well as the execution time (in seconds on four processors) are given for several JD implementations with $it_{SOL} = 1$ *and* $\varepsilon_{tr} = 1.0$.

	Number of JD steps to find first eigenvalue					
	GMRES(1)	GMRES(2)	GMRES(5)	GMRES(10)	GMRES(15)	GMRES(20)
JD-sol	23	25	26	30	32	26
JD-sol*	23	24	21	21	13	12
JD-solP	23	33	13	19	22	28
JD-solP*	23	28	15	12	12	12
	Execution time to find first eigenvalue					
JD-sol	1.68	2.55	4.89	10.09	15.58	16.38
JD-sol*	1.44	1.97	2.69	4.51	2.33	2.28
JD-solP	1.69	3.46	2.31	6.20	10.49	17.82
JD-solP*	1.45	2.46	1.49	1.36	1.76	2.29
	Number of JD steps to find 15 eigenvalues					
JD-sol	72	139	140	109	107	82
JD-sol*	72	105	122	100	92	70
JD-solP	74	188	117	97	79	71
JD-solP*	75	147	119	90	71	55
	Execution time to find 15 eigenvalues					
JD-sol	7.19	17.61	29.82	39.25	54.80	54.53
JD-sol*	6.87	12.94	24.98	33.77	43.61	40.43
JD-solP	7.43	23.74	25.08	35.20	40.63	47.26
JD-solP*	7.26	18.15	24.33	30.25	31.83	30.45

first steps of the process the vectors v_k are usually poor approximations of the wanted eigenvectors, and solving the correction equation more accurately will not speed up the convergence. Therefore, we take in the first steps $it_{SOL} = 0$ until the 2-norm of the actual residual $\hat{r}$ in (7) drops below a threshold value ε_{tr}, i.e., for $\varepsilon_{tr} = 1.0$ after 9 "Arnoldi" steps then only an additional 3 steps are required to find the first eigenvalue (JD-sol*, GMRES(20)). We refer to [6] for more details, discussions and variants of the JD method and the influence of the way the correction equation is solved on the convergence behavior of the JD method.

If the correction equation is solved by GMRES(m) preconditioned from the left by $\mathcal{M}^{-1}$, then in almost all cases the number of JD iterations to the first eigenvalue is smaller than when no preconditioning is applied. This is also true for the complete process of finding 15 eigenvalues, when sufficient GMRES inner iterations are applied. Compared to a multiplication with $Q - \theta I$, the application of $\mathcal{M}^{-1}$ is relatively cheap and as a consequence, the P-variants are usually much faster than those without preconditioning. The convergence history of JD-solP* with GMRES(20) is also displayed in Figure 4. Notice that the residual after an eigenpair is accepted in case $it_{SOL} = 0$ remains much smaller ($\approx 10^{-4}$) compared to the JD-solP*-variant.

However, each GMRES(m) step requires a multiplication with $Q - \theta I$, a very expensive operation even when performed in parallel. As a consequence, a reduction in the number of JD steps due to applying GMRES(m) does not guarantee a reduction in execution time. In fact, the timings listed in Table 3 indicate that GMRES(1) provides the fastest way to find 15 eigenvalues. When we replace step 4 in the algorithm of Figure 2 by $\tilde{z} = r$ (corresponding to $it_{SOL} = 0$) the number of JD steps hardly increases, resulting in the lowest execution time (see Table 4). Based on these and other results (not included in this paper), we continue with JD with $it_{SOL} = 0$.

TABLE 4
Both the number of Jacobi-Davidson iterations as well as the execution time (in seconds on four processors) is given for JD with $it_{SOL} = 0$.

	1 eigenvalue		15 eigenvalues	
	# JD steps	Time	# JD steps	Time
JD, $it_{SOL} = 0$	23	0.99	78	5.23

5.2. Parallel performance on the Cray T3E. In the rest of this paper, we set $it_{SOL} = 0$ which implies that $\tilde{z} = r$ (cf. Table 2, step 4). From numerical experiments this appears to be the best choice for minimizing the total wall clock time, as is illustrated in section 5.1. Three generalized eigenvalue problems, exploited by the linearized MHD equations (7)–(10) in [17] with resistivity $\eta = 5 \cdot 10^{-6}$ and ratio of specific heats $\gamma = \frac{5}{3}$, generated by CASTOR have been timed. Their specifications are given in Table 5. N is the number of diagonal block and the number of Fourier

TABLE 5
The MHD problems used in the numerical experiments.

Problem	N	n	σ	$nnz(A)$	$nnz(B)$
I	40	64	(-0.15, 0.70)	196300	89196
II	160	64	(-0.20, 0.70)	798632	362991
III	320	128	(-0.20, 0.70)	6491568	2886495

modes is equal to $n/16$, where n denotes the blocksize. All problems describe the same physical problem. In [17] more information on the background concerning resistive magnetohydrodynamic spectra in tokomaks can be found. The other parameters in the algorithm were chosen as follows:

$$iter = 300; \; N_{ev} = 15; \; tol_{JD} = 10^{-6}; \; k_{min} = 10; \; m = 30.$$

Table 6 shows the results for the test problems I, II, and III. The third column shows the wall clock time (measured in seconds) required for the construction of the parallel block-tridiagonal LU decomposition. The numbers in parentheses show the Mflop rates, calculated by using the estimate (14) for the number of multiplications. For the block-tridiagonal LU decomposition we obtain a reasonable fraction of the theoretical peak performance, which is due to the level-3 BLAS routines that perform most of the work in this preprocessing phase. Problem I shows that when the number of processors increases from one to two, the wall clock time increases, also. This is due to the computation of the extra fill-in blocks in L and U which are generated when the reordering technique for parallel processing is performed. Increasing the number of processors again by a factor two approximately halves the wall clock time. The reduction in wall clock time for $p = 40$ compared with $p = 20$ is very poor due to the fact that pure cyclic reduction has been applied, since $N = p$. When we calculate those speed-ups for Problems II and III, we see that a gain of 1.44 and 1.65 is reached, respectively. From the performance of more than 10 Gflop/s on 64 processors achieved for the largest problem, we may conclude that the block-tridiagonal LU decomposition is extremely suitable for this platform. However, we must realize that for growing block size the decomposition becomes very expensive.

TABLE 6
Wall clock times (in seconds) of several parts of the JD process on the Cray T3E for Problems I, II, *and* III.

Problem	p	Time for the LU decomposition	Time for JD process	Number of JD steps	Sequential time	Total time triangular solves
I	1	1.06 (184)	16.99	81	0.77	4.74 (67)
I	2	1.15 (315)	10.07	85	0.81	3.92 (114)
I	4	0.60 (750)	5.74	87	0.82	2.11 (244)
I	5	0.52 (899)	4.78	80	0.78	1.76 (275)
I	8	0.34 (1440)	3.62	84	0.85	1.23 (424)
I	10	0.32 (1542)	3.23	79	0.74	1.17 (425)
I	20	0.26 (2003)	2.70	79	0.72	1.02 (495)
I	40	0.24 (2155)	2.44	71	0.76	0.91 (508)
II	2	4.85 (300)	153.60	300	3.77	57.98 (109)
II	4	2.44 (732)	72.59	300	3.77	29.29 (242)
II	5	2.00 (930)	59.40	300	3.76	24.17 (299)
II	8	1.27 (1547)	30.14	247	3.08	12.69 (485)
II	10	1.07 (1864)	25.23	241	3.01	10.62 (571)
II	16	0.70 (2908)	21.32	300	3.74	8.84 (868)
II	20	0.62 (3305)	21.08	300	4.61	8.18 (942)
II	32	0.45 (4654)	16.13	300	4.08	6.06 (1282)
II	40	0.43 (4857)	10.26	198	2.41	4.01 (1281)
III	10	13.52 (2355)	184.07	300	3.49	95.88 (630)
III	16	8.81 (3708)	109.82	300	3.58	61.91 (991)
III	20	7.28 (4525)	92.13	300	3.38	53.64 (1149)
III	32	4.97 (6700)	39.38	196	2.19	23.87 (1700)
III	40	4.42 (7565)	54.45	300	3.53	33.80 (1842)
III	64	3.22 (10461)	39.50	300	3.59	25.41 (2461)

The fourth column shows the wall clock time required for the JD process without the time required for preprocessing. Since a part of the algorithm is not performed in parallel (solution of the projected eigenvalue problem), we cannot expect linear speed-up. The fifth column gives the number of Jacobi–Davidson iteration steps. If this number is equal to 300, less than 15 eigenpairs have been found. In those cases it should have been better to choose a smaller value for N_{ev}. For Problem II approximately 54 iterations were needed to obtain 14 eigenpairs. We emphasize that it is hard to predict how many eigenvalues can be found in the neighborhood of some target σ. It appears that the first 12 eigenvalues of Problem III are easy to find within 100 iterations, whereas 300 iterations are not sufficient to get 15 of them.

The sixth column displays the time spent by solving the eigenvalue problem on the projected system. For an equal number of iterations, this time should be independent of the number of processors involved. However, we found deviating values for $p = 20$ and $p = 32$ in the case of Problem II. We observe that the contribution of the

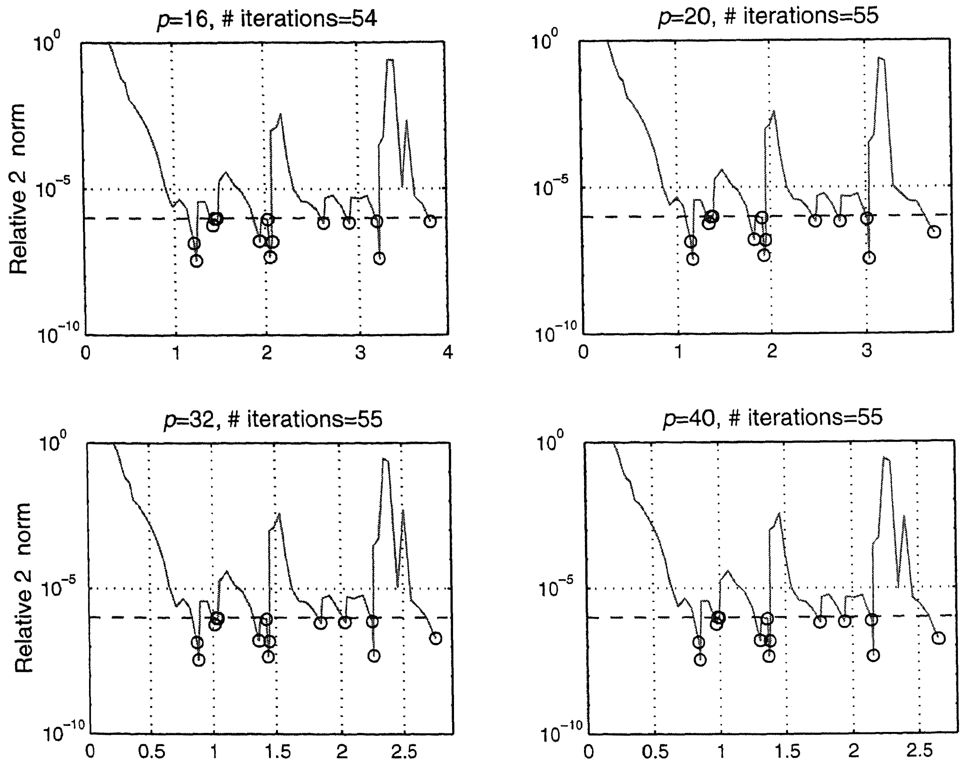

FIG. 5. *The relative* 2*-norm of the residual* (7) *or* (8) *against the wall clock times (in seconds) of Problem* II *(first* 14 *eigenpairs) obtained on* $p = 16, 20, 32, 40$ *processors of the Cray T3E.*

sequential part to the total wall clock time is small, especially for large $\lceil \frac{N}{p} \rceil$.

The seventh column shows the total time spent in performing the triangular solves. Again the numbers in parentheses show the Mflop rates, which have been calculated by using the estimate (15) of the number of multiplications. The level-2 BLAS routines used for the triangular solves are significantly slower than the level-3 BLAS routines used for the construction of L and U, because the ratio of computations and memory-to-processor data transfer is much more favorable for level-3 BLAS than for level-2 BLAS. Therefore, the Mflop rates are significantly lower than those obtained for the construction of L and U. The triangular solves in combination with one matrix-vector multiplication with the matrix B form the expensive application of the operator $Q = (LU)^{-1}B$.

In general, the convergence history hardly depends on the number of processors. This is illustrated by Figure 5, which shows the history of Problem II for the first 14 eigenpairs on 16, 20, 32, and 40 processors. In all cases, the"same" eigenvalues were found in the same order. The distance between the "same" eigenvalues is about 10^{-6} as we might expect, since $tol_{JD} = 10^{-6}$.

5.3. Analysis of the speed-ups. The computations in the JD algorithm can be divided into three parts: the solution of the (small) projected eigenvalue problems, the triangular solves with L and U, and a part consisting of matrix-vector multiplications with B and $A - \sigma B$, inner products and vector updates. The CPU times per processor for these parts are denoted by $t_{seq}^{(p)}$, $t_{LU}^{(p)}$, and $t_{lp}^{(p)}$, respectively; in case p processors are used for execution. If we assume that the time for communication can be neglected and the inner products scale linearly, the expected speed-up $S_{JD}^{(p)}$ on p processors is

$$S_{JD}^{(p)} = \frac{t_{seq}^{(1)} + t_{LU}^{(1)} + t_{lp}^{(1)}}{t_{seq}^{(p)} + \frac{t_{LU}^{(p)}}{S_{LU}^{(p)}} + \frac{t_{lp}^{(p)}}{p}}, \tag{19}$$

in which $S_{LU}^{(p)}$ is the expected speed-up for the triangular solves. In the following, we will show that an approximation of $S_{LU}^{(p)}$, $p > 1$, is given by ($N_p = \lceil \frac{N}{p} \rceil$)

$$\begin{array}{ll} S_{LU}^{(p)} \approx \frac{3pN}{(5-\frac{2}{p})(N+1-p)+5p\lceil \log_2 p - 1\rceil} & \text{for } N_p > 1, \\ S_{LU}^{(p)} \approx \frac{3p}{5\lfloor \log_2 p \rfloor} & \text{for } N_p = 1. \end{array} \tag{20}$$

If the JD code executes on one processor, the sequential block-tridiagonal LU approach is applied and in that case the number of multiplications in the triangular solves is approximately $3Nn^2$. Almost every multiplication can be combined with one addition, hence the wall clock time required on one processor is approximately $3Nn^2 t_{mul}$, in which t_{mul} is the time required for performing a complex multiplication combined with an addition. On p processors, the parallel direct solver (SOL)DDCR is applied. The first part of the triangular solves corresponds to the solution of $N-p+1$ block-tridiagonal systems, which scales linearly. The number of multiplications in this part is approximately $3(N-p+1)n^2\omega$, in which ω is the number of multiplications in the DDCR approach divided by the number of multiplications in the block-tridiagonal LU approach. From (15) it follows that $\omega \approx (5 - \frac{2}{p})/3$. Hence the wall clock time required for the domain decomposition part is approximately $(3\omega(N-p+1)n^2 t_{mul})/p$.

The second part deals with cyclic reduction. The wall clock time required for this part is approximately $5n^2 t_{mul}$ multiplied by the number of steps in cyclic reduction. In case $N_p > 1$, $p-1$ processors perform the cyclic reduction in $\lceil \log_2 p - 1 \rceil$ steps. Otherwise, in the case of pure cyclic reduction ($N_p = 1$), the process is performed on p processors in $\lfloor \log_2 p \rfloor$ steps. The expressions in (20) are obtained by dividing the approximate wall clock time on one processor by the sum of the wall clock times required for the first and second part of the triangular solves on p processors.

Figure 6 shows both the predicted speed-ups by (19) and (20) and the measured speed-ups obtained from Table 6. Since this table does not contain the information for $p = 1$ for Problems II and III, the execution time must be approximated by, for instance,

$$T^{(1)} = t_{seq}^{(p)} + S_{LU}^{(p)} \times t_{LU}^{(p)} + p \times t_{lp}^{(p)} \quad \text{for both } p = 2 \text{ (Problem II) and } p = 10 \text{ (Problem III)}. \tag{21}$$

The large variation in the number of JD steps has been balanced out by dividing $t_{seq}^{(p)}$, $t_{LU}^{(p)}$, and $t_{lp}^{(p)}$ by those numbers. For Problem I the predicted speed-up $S_{JD}^{(p)}$ is a little too high for p large compared with the measured speed-up. This is caused by the fact that the simple prediction model assumes that the inner products scale linearly, which is not quite true for small vectors.

Notice that the speed-ups become higher when the problem size grows, which could be expected. If we increase the order of A and B, $t_{seq}^{(p)}$ hardly changes, when we keep the order of the projected systems fixed. Accordingly, the communication time due to inner products remains equal. However, the computational work expressed by $t_{LU}^{(p)}$ and $t_{lp}^{(p)}$ will increase with the order of the eigenvalue problem.

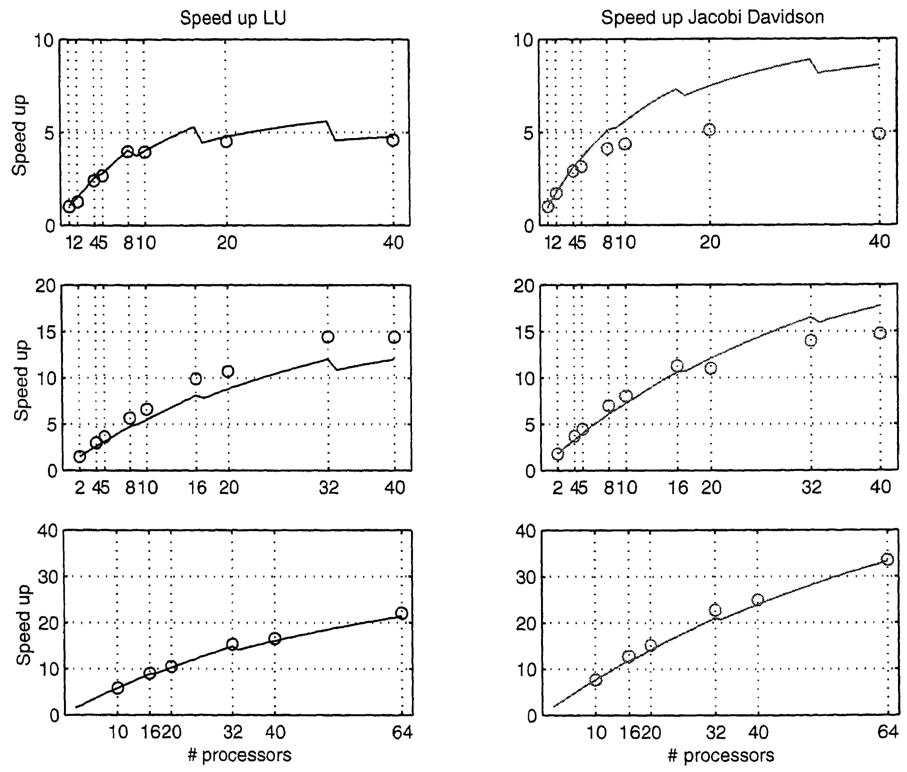

FIG. 6. *The predicted (solid line) and measured speed-up ("o") of the block-tridiagonal LU factorization and the JD algorithm for Problems* I, II, *and* III, *respectively.*

6. Conclusions. We have studied the JD method for the parallel computation of a few selected eigenvalues of large generalized eigenvalue problems arising in the stability investigation of tokamak plasmas. The method has been combined successfully with a parallel complete block-tridiagonal LU decomposition, which appears to be robust because pivoting is used. However, in order not to disturb the block structure of the matrix, the search for pivot elements is restricted to the blocks on the main diagonal. Numerical experiments performed on a Cray T3E demonstrate that this method parallelizes well.

Most other ingredients in the JD method like the matrix-vector multiplications, vector updates and inner products parallelize very well. Only the construction and solution of the small projected eigenvalue problems in the JD method do not parallelize. However, the order of these systems is kept small and is independent of the problem size. Hence for large applications, the total time spent in solving the projected systems is small compared with the time spent in the parallel parts of the method.

We observe that for large problems the measured speed-up corresponds quite well with the prediction. More important is that the speed-up for large problems is much higher than for smaller ones; therefore we may conclude that the JD method is very well suited for parallel execution.

The main reason to study implementations on distributed memory machines, like the Cray T3E for large eigenvalue problems, is the memory bound of shared memory machines. If n and N are large ($n \gg m$ and $N \gg p$), then our parallel implementation of JD requires approximately $128Nn^2$ bytes of memory. Our sequential implementation requires approximately $92Nn^2$ bytes. The total amount of memory of the Cray T3E at HPαC, Delft, is only slightly larger than the 8 Gbytes of main memory of the Dutch National supercomputer with shared memory, a Cray C90 with 12 CPUs at SARA, Amsterdam. Hence on the current Delft configuration, we can solve problems of approximately the same size on the Cray C90. Recently, we got the opportunity to execute our code on a 512 processor Cray T3E, which enables us to examine larger eigenvalue problems (see also [12]).

We remark that the JD method is applicable for generalized eigenvalue problems $Ax = \lambda Bx$ in which the action of the inverse of A, B, or $A - \sigma B$ for a target σ is not available. The projections included in the correction equation guarantee a proper update of an approximate eigenvector in the "right" direction. When we started our research, we hoped to be able to exploit the sparsity pattern within the block-tridiagonal structure of $A - \sigma B$, by using an *incomplete* LU decomposition as a preconditioner for `GMRES(m)` to solve the correction equation approximately. If an *incomplete* decomposition would have been used, the transformation described by (2) would not have been possible and in that case the application of JD applied to the *generalized* eigenvalue problem would probably have been more efficient than Arnoldi's method. However, it turned out to be more efficient to construct a *complete* block-tridiagonal LU decomposition of $A - \sigma B$, because the block tridiagonal structure of this matrix can be exploited. Moreover, since the decomposition is performed on a block level, `BLAS` routines can be used, which guarantees an efficient implementation per processor. We therefore end up with the *standard* eigenvalue problem (2) in which the spectrum of interest is also the dominant part of the spectrum.

For this special eigenvalue problem it is not efficient to use several `GMRES` steps for the correction equation that appears in the JD method. Four variants have been examined and the best results were obtained when the iteration process starts with some Arnoldi-like steps, until the residual is sufficiently small. The process can be accelerated by applying a (cheap) preconditioner within `GMRES(m)`. However, a reduction in the number of Jacobi–Davidson steps does not guarantee a reduction in execution time. From numerical experiments, it appears that the best choice for minimizing the wall clock time is to take $\tilde{z} = r$ in step 4 of the algorithm in Figure 2. If we should use a (very) good preconditioner and solve the correction equation exactly, then the convergence to the next eigenvector will be quadratic (see [15, Theorem 3.2]). For this special case, it is probably more efficient to use Arnoldi's method than JD. However, the difference in efficiency will not be large, because the most expensive operation in both JD and Arnoldi's method is the matrix-vector multiplication with Q in (2) and both methods need approximately the same number of matrix-vector multiplications.

Acknowledgments. The authors wish to thank Herman te Riele for many stimulating discussions and numerous suggestions for improving the presentation of the paper. They also thank Gerard Sleijpen and Jos van Dorsselaer for carefully reading the paper and suggesting several improvements. We also appreciate the helpful comments by the referees very much.

Further they would like to thank Ronald van Pelt of Cray Research, the Netherlands, for his programming advice for both the Cray T3D and Cray T3E, and HPαC (Delft) and SARA (Amsterdam) for their technical support. They gratefully acknowledge Cray Research for a sponsored account on the Cray T3D, and the Dutch National Computing Facilities Foundation NCF for the provision of computer time on the Cray

C90 and the Cray T3E.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J.J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*, 2nd ed., SIAM, Philadelphia, 1995.

[2] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994; also available online from http://www.netlib.org/templates/.

[3] J.G.C. Booten and H.A. van der Vorst, *Cracking large-scale eigenvalue computations, part I: Algorithms*, Computers in Physics, 10 (1996), pp. 239–242.

[4] J.G.C. Booten and H.A. van der Vorst, *Cracking large-scale eigenvalue computations, Part II: Implementations*, Computers in Physics, 10 (1996), pp. 331–334.

[5] J.G.C. Booten, D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst, *Jacobi-Davidson methods for generalized MHD-eigenvalue problems*, Z. Angew. Math. Mech. 76, Supplement 1 (1996), pp. 131–134.

[6] D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.

[7] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[8] W. Kerner, J.P. Goedbloed, G.T.A. Huysmans, S. Poedts, and E. Schwartz, *CASTOR: Normal-mode analysis of resistive MHD plasmas*, J. Comput. Physics, 142 (1998), pp. 271–303.

[9] W. Kerner, S. Poedts, J.P. Goedbloed, G.T.A. Huysmans, B. Keegan, and E. Schwartz, *Computing the damping and destabilization of global Alfvén waves in tokamaks*, in Proceedings of 18th Conference on Controlled Fusion and Plasma Physics, Vol. IV, P. Bachman and D. C. Robinson, eds., EPS, Berlin, 1991, pp. 89–92.

[10] R.B. Lehoucq, D.C. Sorensen, and C. Yang, *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.

[11] M. Nool and A. van der Ploeg, *A Parallel Jacobi-Davidson Method for Solving Generalized Eigenvalue Problems in Linear Magnetohydrodynamics*, Technical Report NM-R9733, CWI, Amsterdam, 1997.

[12] M. Nool and A. van der Ploeg, *Parallel Jacobi-Davidson for solving generalized eigenvalue problems*, in Proceedings of the Third International Meeting on Vector and Parallel Processing, Lecture Notes in Comput. Sci. 1573, J.M.L.M. Palma, J. Dongarra, and V. Hernandez, eds., Springer-Verlag, Berlin, 1999, pp. 58–71.

[13] S. Poedts, W. Kerner, J.P. Goedbloed, B. Keegan, G.T.A. Huysmans, and E. Schwartz, *Damping of global Alfvén waves in tokamaks due to resonant absorption*, Plasma Physics and Controlled Fusion, 34 (1992), pp. 1397–1422.

[14] Y. Saad and M.H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 17 (1986), pp. 856–869.

[15] G.L.G. Sleijpen, J.G.L. Booten, D.R. Fokkema, and H.A. van der Vorst, *Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems*, BIT, 36 (1996), pp. 595–633.

[16] G.L.G Sleijpen and H.A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.

[17] B. van der Holst, A.J.C. Beliën, J.P. Goedbloed, M. Nool, and A. van der Ploeg, *Calculation of resistive magnetohydrodynamics spectra in tokamaks*, Physics of Plasmas, 6 (1999), pp. 1554–1561.

[18] A. van der Ploeg, *Reordering Strategies and LU-Decomposition of Block Tridiagonal Matrices for Parallel Processing*, Technical Report NM-R9618, CWI, Amsterdam, 1996.